MINISTÈRE
DE L'ÉDUCATION
NATIONALE

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

EBE MAT 1

SESSION 2018

CAPES
CONCOURS EXTERNE
ET CAFEP

Sections:

MATHÉMATIQUES
LANGUES RÉGIONALES : BRETON

PREMIÈRE ÉPREUVE D'ADMISSIBILITÉ

Durée : 5 heures

Option Mathématiques :

Calculatrice électronique de poche - y compris calculatrice programmable, alphanumérique ou à écran graphique – à fonctionnement autonome, non imprimante, autorisée conformément à la circulaire n° 99-186 du 16 novembre 1999.
L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Option Informatique :

Calculatrice électronique de poche - y compris calculatrice programmable, alphanumérique ou à écran graphique – à fonctionnement autonome, non imprimante, autorisée conformément à la circulaire n° 99-186 du 16 novembre 1999.
L'usage de tout ouvrage de référence, de tout dictionnaire et de tout autre matériel électronique est rigoureusement interdit.

Les candidats doivent traiter le sujet de l'option choisie lors de l'inscription.

Dans le cas où un(e) candidat(e) repère ce qui lui semble être une erreur d'énoncé, il (elle) le signale très lisiblement sur sa copie, propose la correction et poursuit l'épreuve en conséquence.

De même, si cela le (la) conduit à formuler une ou plusieurs hypothèses, il lui est demandé de la (ou les) mentionner explicitement.

NB : ***La copie que vous rendrez ne devra, conformément au principe d'anonymat, comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé comporte la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de signer ou de porter quelque signe d'identification que ce soit.***

A

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

► Concours externe du CAPES de l'enseignement public :

- **option Mathématiques** :

Concours	Section/option	Epreuve	Matière
EBE	13OOE	101	0466

- **option Informatique** :

Concours	Section/option	Epreuve	Matière
EBE	13OOE	101	0413

- **Langue régionale Breton** :

Concours	Section/option	Epreuve	Matière
EBE	0441E	102	0466

► Concours externe du CAFEP/CAPES de l'enseignement privé :

- **option Mathématiques:**

Concours	Section/option	Epreuve	Matière
EBF	13OOE	101	0466

- **option Informatique** :

Concours	Section/option	Epreuve	Matière
EBF	13OOE	101	0413

- **Langue régionale Breton** :

Concours	Section/option	Epreuve	Matière
EBF	0441E	102	0466

Option Mathématiques

Le sujet est comporte cinq parties.

Notations
$\mathbb{N}$ désigne l'ensemble des entiers naturels et $\mathbb{N}^*$ l'ensemble des entiers naturels non nuls.
Pour m et n deux entiers naturels, $[\![m;n]\!]$ désigne l'ensemble des entiers k tels que $m \leqslant k \leqslant n$.
$\mathbb{Z}$ désigne l'ensemble des entiers relatifs.
$\mathbb{Q}$ désigne l'ensemble des nombres rationnels.
$\mathbb{R}$ désigne l'ensemble des nombres réels.
On note e le nombre $\exp(1)$, image de 1 par la fonction exponentielle.
On rappelle que, pour tout nombre réel x, il existe un unique entier relatif $E(x)$ tel que $E(x) \leqslant x < E(x)+1$. Cet entier $E(x)$ est appelé *partie entière de* x.

Partie A : suites adjacentes

Étant donné deux suites réelles $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$, on rappelle qu'elle sont dites *adjacentes* si l'une des deux est croissante, l'autre décroissante et si $\lim\limits_{n\to+\infty}(a_n - b_n) = 0$.

I. On suppose dans cette question que la suite $(a_n)_{n\in\mathbb{N}}$ est croissante et que la suite $(b_n)_{n\in\mathbb{N}}$ est décroissante.

1. Montrer que la suite $(a_n - b_n)_{n\in\mathbb{N}}$ est monotone et en déduire que pour tout entier naturel n, $a_n \leqslant b_n$.

2. Justifier que les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ sont convergentes vers une même limite ℓ vérifiant :

$$\forall n \in \mathbb{N}, \quad a_n \leqslant \ell \leqslant b_n.$$

3. On suppose de plus les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ strictement monotones. Montrer que :

$$\forall n \in \mathbb{N}, \quad a_n < \ell < b_n.$$

II. Pour tout entier naturel n non nul, on pose $a_n = \sum\limits_{p=0}^{n} \dfrac{1}{p!}$ et $b_n = a_n + \dfrac{1}{n \times n!}$.

1. Montrer que les suites $(a_n)_{n\in\mathbb{N}^*}$ et $(b_n)_{n\in\mathbb{N}^*}$ sont adjacentes.

2. Démontrer que pour tout entier naturel n non nul, $\mathrm{e} - a_n = \dfrac{1}{n!}\displaystyle\int_0^1 (1-t)^n \mathrm{e}^t \mathrm{d}t$.
Indication : on pourra procéder par récurrence.

3. En déduire que pour tout entier naturel n non nul, $0 < \mathrm{e} - a_n < \dfrac{1}{n \times n!}$.
En déduire la limite de la suite $(a_n)_{n\in\mathbb{N}^*}$.
Indication : on pourra étudier les variation de la fonction $t \mapsto (1-t)e^t$.

4. En déduire une valeur de n telle que a_n soit une valeur approchée de e à 10^{-5} près.

 Tournez la page S.V.P.

5. On suppose que e est un nombre rationnel.

a. Montrer qu'il existe un entier naturel non nul q tel que le nombre $e\,q!$ soit un entier naturel.

b. Montrer que $x = q!\left(e - \sum_{p=0}^{q} \frac{1}{p!}\right)$ est un entier naturel.

c. Montrer que $0 < x < 1$.

d. Conclure.

Soit f une fonction à valeurs réelles définie sur un intervalle ouvert I contenant 0. On rappelle que f est dite *développable en série entière* au voisinage de 0 s'il existe un nombre réel $R > 0$ et une suite $(a_n)_{n\geqslant 0}$ de nombres réels tels que $]-R, R[$ est inclus dans I et :

$$\forall x \in]-R, R[, \quad f(x) = \sum_{k=0}^{+\infty} a_n x^n.$$

III. **1.** Démontrer que la fonction $x \mapsto \dfrac{1}{1+x}$ est développable en série entière au voisinage de 0. Préciser son développement et donner le rayon de convergence de cette série entière.

2. Justifier que, pour tout nombre réel x dans l'intervalle $]-1, 1[$,

$$\ln(1+x) = \sum_{k=0}^{+\infty} (-1)^k \frac{x^{k+1}}{k+1}.$$

On énoncera avec soin le théorème utilisé.

3. Pour $x \in [0,1]$ et $n \in \mathbb{N}$, on pose $S_n(x) = \sum_{k=0}^{n} (-1)^k \dfrac{x^{k+1}}{k+1}$.

Démontrer que les deux suites $(S_{2n}(x))_{n\in\mathbb{N}}$ et $(S_{2n+1}(x))_{n\in\mathbb{N}}$ sont adjacentes.

4. En déduire que, pour tout entier naturel n et tout nombre réel x dans l'intervalle $[0, 1[$,

$$S_{2n+1}(x) \leqslant \ln(1+x) \leqslant S_{2n}(x)$$

5. En déduire que, pour tout entier naturel n,

$$S_{2n+1}(1) \leqslant \ln(2) \leqslant S_{2n}(1).$$

6. Démontrer que $\ln(2) = \sum_{k=0}^{+\infty} \dfrac{(-1)^k}{k+1}$.

Partie B : écriture d'un entier en base deux

Le but de cette partie est de démontrer que tout entier naturel N supérieur ou égal à 2 s'écrit de manière unique

$$N = \sum_{k=0}^{n-1} d_k 2^k \quad \text{avec} \quad n \geqslant 2 \text{ et } \begin{cases} \forall k \in [\![0\ ;\ n-2]\!], \quad d_k \in \{0,1\}, \\ d_{n-1} = 1. \end{cases}$$

L'égalité précédente se note $N = \overline{d_{n-1} d_{n-2} \ldots d_0}$ (écriture de N en base deux) ; la suite finie $(d_k)_{0\leqslant k\leqslant n-1}$ s'appelle la suite des chiffres dans l'écriture de N en base deux.

Dans toute cette partie, N désigne un entier naturel supérieur ou égal 2.

IV. On suppose que $N = \sum_{k=0}^{n-1} d_k 2^k$ avec $\forall k \in [\![0\ ;\ n-2]\!]\quad d_k \in \{0,1\}$ et $d_{n-1} = 1$.

1. Montrer que $2^{n-1} \leqslant N \leqslant 2^n - 1$.

2. Montrer que d_0 est le reste de la division euclidienne de N par 2.

3. Démontrer que la suite $(d_0, \ldots, d_{n-1})$ est déterminée de manière unique.

V. On définit deux suites d'entiers $(y_k)_{k\in\mathbb{N}}$ et $(d_k)_{k\in\mathbb{N}}$ par $y_0 = N$ et pour tout entier naturel k, y_{k+1} et d_k désignent respectivement le quotient et le reste de la division euclidienne de y_k par 2.

1. On fixe $k \in \mathbb{N}^*$. Exprimer N en fonction de k, $d_0, \ldots, d_{k-1}$ et y_k.

2. Démontrer que la suite $(y_k)_{k\in\mathbb{N}}$ est nulle à partir d'un certain rang et qu'il existe un entier $n \geqslant 1$ tel que $\overline{d_{n-1}d_{n-2}\ldots d_0}$ soit l'écriture de N en base deux.

3. Écrire un algorithme qui, pour tout entier naturel N supérieur ou égal 2 donné, renvoie la suite $(d_0, d_1, \ldots, d_{n-1})$ des chiffres de son écriture en base deux.

4. Écrire en base deux le nombre qui s'écrit 391 en base dix.

VI. On se propose à présent de calculer le nombre N qui s'écrit $\overline{d_{n-1}d_{n-2}\ldots d_0}$ en base deux.

1. Première méthode : méthode « naïve ».
On écrit $N = \sum_{k=0}^{n-1} d_k 2^k$. Combien d'opérations (additions et multiplications) doit-on effectuer a priori pour calculer N avec cette méthode ?

2. Deuxième méthode : méthode de Hörner.
On écrit $N = ((((d_{n-1} \times 2 + d_{n-2}) \times 2 + d_{n-3}) \times 2 + \ldots) \times 2 + d_0$. Combien d'opérations (additions et multiplications) doit-on effectuer a priori pour calculer N avec cette méthode ?

3. Écrire un algorithme qui, pour toute suite de chiffres $(d_0, \ldots, d_{n-1})$ donnée, renvoie la valeur de N calculée à l'aide de cette deuxième méthode.

4. Quel est le nombre dont l'écriture en base deux est $\overline{101001000100001}$?

Partie C : nombres dyadiques

L'ensemble $D_2 = \left\{ \frac{a}{2^p}\ ;\ a \in \mathbb{Z},\ p \in \mathbb{N} \right\}$ est appelé ensemble des nombres dyadiques. On note D_2^+ l'ensemble des nombres dyadiques positifs ou nuls.

VII. Montrer que $\mathbb{Z}$ est strictement inclus dans D_2 et que D_2 est strictement inclus dans $\mathbb{Q}$.
Indication : on pourra montrer que $\frac{1}{3} \notin D_2$.

VIII. Soit $x \in D_2^+ \backslash \mathbb{N}$. On se propose de démontrer qu'il existe un unique entier $n \geqslant 1$ et une unique suite $(a_0, a_1, \ldots, a_n)$ avec $a_0 \in \mathbb{N}$ et $(a_1, \ldots, a_n) \in \{0,1\}^n$ tels que

$$x = \sum_{k=0}^{n} a_k 2^{-k}, \quad \text{avec } a_n \neq 0.$$

Le membre de droite de cette égalité s'appelle le développement dyadique de x.

1. On suppose qu'une telle suite existe. Montrer que $a_0 = E(x)$ puis montrer que la suite $(a_0, a_1, \ldots, a_n)$ est déterminée de manière unique.

Tournez la page S.V.P.

2. On souhaite à présent montrer l'existence d'une telle suite. À l'aide de la partie précédente, montrer l'existence d'un entier a_0, d'un entier $p \geqslant 1$ et d'une suite de nombres entiers $d_0, \ldots, d_{p-1}$ égaux à 0 ou 1, non tous nuls, tels que

$$x = a_0 + \sum_{k=0}^{p-1} d_k 2^{k-p}.$$

3. Conclure.

IX. Donner le développement dyadique de $\dfrac{35}{4}$.

Partie D : développement dyadique illimité

On appelle suite dyadique toute suite $(a_k)_{k\in\mathbb{N}*}$ où pour tout $k \in \mathbb{N}^*$, a_k est un élément de $\{0, 1\}$. De plus :

- une suite dyadique $(a_k)_{k\in\mathbb{N}*}$ est dite impropre s'il existe un entier $m \in \mathbb{N}^*$ tel que pour tout $k \geqslant m$, $a_k = 1$;
- une suite dyadique $(a_k)_{k\in\mathbb{N}*}$ est dite propre si elle n'est pas impropre.

X. On suppose que $a = (a_k)_{k\in\mathbb{N}*}$ est une suite dyadique.

1. Démontrer que la série de terme général $a_k 2^{-k}$ est convergente. On note sa somme $s(a) = \sum_{k=1}^{+\infty} a_k 2^{-k}$.

2. Soit N un entier naturel. Que vaut $\sum_{k=N}^{+\infty} 2^{-k}$?

3. Vérifier que $s(a) \in [0, 1]$.

4. Montrer que si a est une suite dyadique propre, alors $s(a) \in [0, 1[$.

5. Montrer que si a est une suite dyadique impropre, alors $s(a)$ est un nombre dyadique.

6. Soit $a = (a_k)_{k\in\mathbb{N}*}$ la suite définie par

$$a_k = \begin{cases} 0 \text{ si } k \text{ est impair,} \\ 1 \text{ si } k \text{ est pair.} \end{cases}$$

Montrer que $s(a) = \dfrac{1}{3}$.

XI. Soit x un nombre dyadique compris dans l'intervalle $[0, 1[$.

1. En utilisant les résultats de la partie C, montrer qu'il existe une suite dyadique propre a telle que

$$x = \sum_{k=1}^{+\infty} a_k 2^{-k}.$$

2. Montrer que si x est non nul, alors il existe également une suite dyadique impropre b telle que

$$x = \sum_{k=1}^{+\infty} b_k 2^{-k}.$$

XII. Dans cette question, on considère un nombre réel x appartenant à l'intervalle $[0,1[$. On lui associe la suite $\alpha(x) = (\alpha_k(x))_{k\in\mathbb{N}^*}$ définie pour tout $k \in \mathbb{N}^*$ par l'égalité

$$\alpha_k(x) = E(2^k x) - 2E(2^{k-1}x).$$

Pour tout $n \in \mathbb{N}^*$, on pose $u_n(x) = \sum_{k=1}^{n} \alpha_k(x)2^{-k}$ et $v_n(x) = u_n(x) + 2^{-n}$.

1. Démontrer que la suite $(\alpha_k(x))_{k\in\mathbb{N}^*}$ est une suite dyadique.

2. Démontrer que les deux suites $(u_n(x))_{n\in\mathbb{N}^*}$ et $(v_n(x))_{n\in\mathbb{N}^*}$ sont adjacentes et prennent leurs valeurs dans $D_2 \cap [0,1]$.

3. Vérifier que $E(2^n x) = 2^n u_n(x)$ et en déduire que pour tout entier naturel $n \geqslant 1$,

$$u_n(x) \leqslant x < v_n(x).$$

4. Quelle est la limite commune des suites $(u_n(x))_{n\in\mathbb{N}^*}$ et $(v_n(x))_{n\in\mathbb{N}^*}$?

5. Montrer que $(\alpha_k(x))_{k\in\mathbb{N}^*}$ est une suite dyadique propre et que

$$x = \sum_{k=1}^{+\infty} \alpha_k(x)2^{-k}.$$

6. En déduire que pour tout nombre réel x dans l'intervalle $[0,1[$, il existe une unique suite dyadique propre $(a_k)_{k\in\mathbb{N}^*}$ telle que :

$$x = \sum_{k=1}^{+\infty} a_k 2^{-k}.$$

On note alors

$$x = \overline{0,a_1a_2a_3\ldots}$$

Cette nouvelle représentation de x est appelée la *représentation dyadique propre* de x. Si la suite $(a_k)_{k\in\mathbb{N}^*}$ est nulle à partir d'un certain rang, on dit que la représentation dyadique de x est finie.

7. Si $d = (d_n)_{n\in\mathbb{N}^*}$ est une suite dyadique propre, on note $x = s(d)$ et $d' = (d_{n+1})_{n\in\mathbb{N}^*}$. Justifier que $d_1 = E(2x)$ et $s(d') = 2x - d_1$.
En déduire un algorithme qui prend en entrées un nombre réel $x \in [0,1[$ et un entier $n \in \mathbb{N}^*$ et qui renvoie la liste des n premiers chiffres du développement dyadique propre de x. On admettra l'existence d'une fonction *floor* qui renvoie la partie entière de son argument.

XIII. Démontrer que $D_2 \cap [0,1]$ est dense dans $[0,1]$. En déduire que D_2 est dense dans $\mathbb{R}$.

XIV. Démontrer que $\mathbb{R}\backslash D_2$ est dense dans $\mathbb{R}$.
Indication : on pourra utiliser la question VII.

XV. Soit x un nombre réel dans l'intervalle $\in [0,1[$ dont un développement dyadique, propre ou impropre, est $\overline{0,a_1a_2a_3\ldots}$

1. Quel est le développement dyadique de $1-x$?

2. On suppose que $2x \in [0,1[$. Quel est le développement dyadique de $2x$? Plus généralement, quel est le développement dyadique de $2^l x$, lorsque l est un entier relatif et que $2^l x \in [0,1[$?

3. Donner le développement dyadique de $\dfrac{2}{3}$.

Tournez la page S.V.P.

Partie E : suite extraite de la suite $(\cos(n\pi\theta))_{n\in\mathbb{N}}$

XVI. Dans cette question, θ désigne un nombre réel strictement positif. On pose

$$c_n = \cos(n\pi\theta), \qquad s_n = \sin(n\pi\theta).$$

1. Vérifier que pour tout entier naturel n,

$$\begin{aligned} c_{n+1} + c_{n-1} &= 2c_n \cos(\pi\theta), \\ c_{n+1} - c_{n-1} &= -2s_n \sin(\pi\theta), \\ c_n^2 + s_n^2 &= 1. \end{aligned}$$

2. En déduire que la suite $(c_n)_{n\in\mathbb{N}}$ converge si et seulement si θ est un entier relatif pair.
Indication : on pourra raisonner par disjonction de cas, suivant la valeur de $\cos(\pi\theta)$.

XVII. On s'intéresse à présent à la suite $(c_{2^n})_{n\in\mathbb{N}}$ extraite de $(c_n)_{n\in\mathbb{N}}$. Pour tout entier naturel n, on pose :

$$u_n = c_{2^n} = \cos(2^n\pi\theta).$$

1. On suppose (dans cette question uniquement) que θ est un nombre dyadique. Quelle est la nature de la suite $(u_n)_{n\in\mathbb{N}}$?

2. On suppose (dans cette question uniquement) qu'il existe un nombre dyadique x tel que $\theta = x + \dfrac{1}{3}$. Quelle est la nature de la suite $(u_n)_{n\in\mathbb{N}}$?

3. On suppose (dans cette question uniquement) qu'il existe un nombre dyadique x tel que $\theta = x + \dfrac{2}{3}$. Quelle est la nature de la suite $(u_n)_{n\in\mathbb{N}}$?

4. Justifier que, pour tout entier naturel n, $u_{n+1} = 2u_n^2 - 1$.

5. Lorsque la suite $(u_n)_{n\in\mathbb{N}}$ converge vers ℓ, quelles sont les seules valeurs possibles pour le réel ℓ ?

6. Soit $(a_n)_{n\in\mathbb{N}^*}$ la suite définissant le développement dyadique propre de $\theta - E(\theta)$. Montrer que, quel que soit l'entier naturel n, il existe un entier relatif k_n et un réel ε_n appartenant à l'intervalle $[0, \frac{1}{2}]$ tels que :

$$2^n\theta = 2k_n + a_n + \frac{a_{n+1}}{2} + \varepsilon_n.$$

7. Démontrer que :
– si $a_n = a_{n+1}$, alors $u_n \geqslant 0$;
– si $a_n \neq a_{n+1}$, alors $u_n \leqslant 0$.
Puis que :
– si $u_n > 0$, alors $a_n = a_{n+1}$;
– si $u_n < 0$, alors $a_n \neq a_{n+1}$.

8. On suppose que la suite $(u_n)_{n\in\mathbb{N}}$ converge vers un nombre réel $\ell > 0$. Montrer qu'à partir d'un certain rang, $a_n = 0$. En déduire que θ est un nombre dyadique.

9. On suppose que la suite $(u_n)_{n\in\mathbb{N}}$ converge vers un nombre réel $\ell < 0$. Montrer qu'à partir d'un certain rang, $a_{n+1} \neq a_n$. En déduire que $\theta - \dfrac{1}{3}$ ou $\theta - \dfrac{2}{3}$ est un nombre dyadique.

XVIII. Énoncer et démontrer une condition nécessaire et suffisante pour que la suite $(u_n)_{n\in\mathbb{N}}$ converge. On justifiera ce résultat et on précisera le cas échéant la valeur de sa limite.

Option Informatique

Le sujet est constitué de deux problèmes indépendants.

Problème n° 1 : suite de Lucas

Rappels et notations. Pour x un nombre réel, il existe un plus grand entier inférieur ou égal à x, appelé *plancher* de x ou *partie entière* de x. Ce nombre entier est noté $\lfloor x \rfloor$. Il existe un plus petit entier supérieur ou égal à x, appelé *plafond* de x.

Les fonctions `ceil` et `floor` du module `math` de Python calculent respectivement le plafond et le plancher d'un nombre flottant x : Ainsi `ceil(1.24)` vaut 2 et `floor(1.24)` vaut 1.

Soit a un nombre réel strictement positif. On désigne par $\log_a$ la fonction logarithme en base a : si x est un nombre réel strictement positif,

$$\log_a x = \frac{\ln x}{\ln a},$$

où ln désigne la fonction logarithme néperien.

Dans ce problème, on étudie plusieurs algorithmes de calcul des nombres de Lucas.

Ces nombres sont les termes de la suite $(L_n)_{n\geqslant 0}$ définie par les relations suivantes :

$$\begin{cases} L_0 = 2, \\ L_1 = 1, \\ L_n = L_{n-1} + L_{n-2} \text{ pour } n \geqslant 2. \end{cases}$$

I. Calculer L_2, L_3, L_4, L_5, L_6 et L_7.

II. Montrer que pour tout $n \geqslant 0$, L_n est un entier naturel.

III. On considère l'équation $x^2 - x - 1 = 0$.

1. Montrer que cette équation possède deux solutions, l'une positive que l'on note ϕ, l'autre négative que l'on note $\hat{\phi}$. Des valeurs approchées à 10^{-4} près de ces deux nombres sont $\phi \approx 1{,}6180$ et $\hat{\phi} \approx -0{,}6180$.

2. Justifier que

$$\phi + 1 = \phi^2, \qquad \hat{\phi} + 1 = \hat{\phi}^2, \qquad \phi + \hat{\phi} = 1, \qquad \phi\hat{\phi} = -1.$$

IV. Montrer que pour tout entier naturel n, $L_n = \phi^n + \hat{\phi}^n$. On pourra raisonner par récurrence.

V. On donne cette valeur approchée du logarithme en base 10 de ϕ :

$$\log_{10} \phi = \frac{\ln \phi}{\ln 10} \approx 0{,}2090.$$

Montrer que pour tout entier naturel p,

$$n \geqslant 5p \Longrightarrow L_n \geqslant 10^p.$$

 Tournez la page S.V.P.

VI. **1.** On propose la fonction suivante pour calculer le n-ième nombre de Lucas. On rappelle que `**` est l'opérateur Python d'élévation à la puissance.

```
0 from math import *
1
2 def lucas1(n):
3     if n == 0:
4         return 2
5     phi = (1+sqrt(5))/2
6     phi2 = (1-sqrt(5))/2
7     return phi**n + phi2**n
```

L'évaluation de `[lucas1(n) for n in range(8)]` renvoie la liste

`[2, 1.0, 3.0, 4.0, 7.000000000000001, 11.000000000000002, 18.000000000000004, 29.000000000000007].`

Pourquoi ne s'agit-il pas d'une liste d'entiers ?

2. On propose maintenant la fonction suivante pour calculer le n-ième nombre de Lucas.

```
0 from math import *
1
2 def lucas2(n):
3     if n == 0:
4         return 2
5     phi = (1+sqrt(5))/2
6     if n%2 == 0:
7         return ceil(phi**n)
8     else:
9         return floor(phi**n)
```

L'évaluation de `[lucas2(n) for n in range(8)]` renvoie la liste

`[2, 1, 3, 4, 7, 11, 18, 29].`

Expliquer le choix de ces fonctions en lignes 6 à 9 et démontrer que, si les calculs en flottants sont exacts, `lucas2(n)` calcule bien L_n.

3. Un calcul exact montre que $L_{36} = 33385282$, mais `lucas2(36)` renvoie la valeur 33385283. Comment expliquez-vous cela ?

VII. On souhaite écrire une nouvelle fonction de calcul des termes de la suite de Lucas. Pour éviter tout problème lié au calcul avec des flottants, on souhaite ne travailler qu'avec des entiers, sur lesquels Python calcule de manière exacte.

1. Recopier et compléter la fonction suivante, qui renvoie la valeur de L_n.

```
0 def lucas3(n):
1     if n == 0:
2         return 2
3     if n == 1:
4         return 1
5     a,b = (2,1)
6     for i in range(n):
7         a,b = (..., ...)
8     return ...
```

2. Déterminer un invariant de boucle qui précise, en fonction de la valeur de `i`, les valeurs des variables `a` et `b` avant et après l'exécution de la ligne 7 et en déduire que `lucas3` renvoie un résultat exact.

3. Pour $n \geqslant 2$, combien d'additions sont effectuées par `lucas3(n)` ?

VIII. **1.** Démontrer que, pour tout entier $n \geqslant 1$,

$$L_n^2 - L_{n+1}L_{n-1} = 5\,(-1)^n.$$

2. Démontrer que, pour tout entier $n \geqslant 1$,

$$\begin{cases} L_{2n} &= L_n^2 - 2(-1)^n, \\ L_{2n+1} &= L_nL_{n+1} - (-1)^n. \end{cases}$$

IX. **1.** Si k est un entier, que calcule l'expression Python suivante : `1 - 2*(k % 2)` ?

On rappelle que l'opérateur `%` calcule le reste dans la division entière : `7 % 3` vaut 1 ; `8 % 3` vaut 2 et `9 % 3` vaut 0.

2. Recopier et compléter la fonction récursive suivante, de sorte que `lucas4(n)` renvoie le couple (L_n, L_{n+1}).

```
0  def lucas4(n):
1      if n == 0:
2          return (2,1)
3      if n == 1:
4          return (1,3)
5      k = n // 2
6      u = 1 - 2*(k % 2)
7      a,b = lucas4(...)
8      if n % 2 == 0:
9          return (..., ...)
10     else:
11         return (..., ...)
```

3. Pour tout entier $n \geqslant 2$, exprimer en fonction de n le nombre d'appels récursifs que réalise `lucas4(n)`.

X. Pour n un entier naturel, on considère le vecteur colonne de $\mathbb{R}^2$ défini par

$$V_n = \begin{pmatrix} L_n \\ L_{n+1} \end{pmatrix}.$$

On considère également la matrice $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Pour tout entier $n \geqslant 1$, exprimer V_n en fonction de A et de V_{n-1}, puis exprimer V_n en fonction de A, de n et de V_0. On représente dans la suite une matrice carrée $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ par la liste de deux listes

`[[a,b], [c,d]]`.

Tournez la page S.V.P.

XI. **1.** Écrire en Python une fonction `prodMat(M1,M2)` qui prend en arguments deux matrices 2×2 représentées par des listes comme indiqué ci-dessus, et qui renvoie la liste qui représente la matrice produit $M1 \times M2$.

2. Combien l'appel `prodMat(M1,M2)` effectue-t-il d'additions ? de multiplications ?

3. On considère la fonction (naïve) d'élévation d'une matrice à une puissance entière suivante.

```
0 def puissanceMat(M,p):
1     R = [[1,0],[0,1]]
2     for i in range(p):
3         R = prodMat(R,M)
4     return R
```

Combien l'appel `puissanceMat(A,p)` réalise-t-il d'appels à `prodMat` en fonction de p ?

XII. L'algorithme d'exponentiation rapide repose sur la remarque que

$$a^{2p+1} = (a^p)^2 a = b \times b \times a,$$

où $b = a^p$, quand $a^{2p} = b \times b$. Selon la parité de p, il faut donc 1 ou 2 multiplications de plus pour calculer a^{2p+1} ou a^{2p} connaissant a^p. Ainsi, pour calculer a^{122}, par exemple :

$$\begin{aligned} a^{122} &= (a^{61})^2 \\ &= (a.(a^{30})^2)^2 \\ &= (a.((a^{15})^2)^2)^2 \\ &= (a.((a.(a^7)^2)^2)^2)^2 \\ &= (a.((a.(a.(a^3)^2)^2)^2)^2)^2 \\ &= (a.((a.(a.(a.a^2)^2)^2)^2)^2)^2. \end{aligned}$$

Ainsi, on se contente de 10 multiplications.

1. On propose la fonction suivante qui implémente l'exponentiation rapide pour les matrices 2×2.

```
0 def puissanceMatRapide(M,n):
1     R = [[1,0], [0,1]]
2     P = M
3     while n > 0:
4         if n%2 == 1:
5             R = prodMat(R,P)
6         P = prodMat(P,P)
7         n = n // 2
8     return R
```

Montrer que par cette méthode, le nombre d'appel à la fonction `prodMat` est majoré par $2 + 2\lfloor \log_2 p \rfloor$.

2. Exprimer en fonction de M et de i la valeur de la matrice P, après la ième itération de la boucle `while`.

3. Soit $n = \overline{c_p \dots c_0}$ l'écriture de n en base 2. Montrer que, pour tout entier i compris entre 1 et p, la valeur de l'entier n après la ième itération de la boucle `while` est donée par

$$n = \sum_{j=i}^{p} c_j 2^{j-i}.$$

4. Montrer que pour tout entier i compris entre 1 et p, la valeur de la matrice R après la ième itération de la boucle `while` est donnée par $R = M^k$, avec

$$k = \sum_{j=0}^{i-1} c_j 2^j.$$

5. Exprimer le nombre d'exécutions de la boucle `while` en fonction de n puis démontrer la correction de l'algorithme proposé, c'est-à-dire que `puissanceMatRapide(M,n)` calcule effectivement la puissance désirée.

XIII. Proposer le code d'une fonction `lucas5(n)` qui retourne la valeur du nombre de Lucas L_n en utilisant la fonction `puissanceMatRapide`.

Tournez la page S.V.P.

Problème n° 2 : emplois du temps et graphes d'intervalles

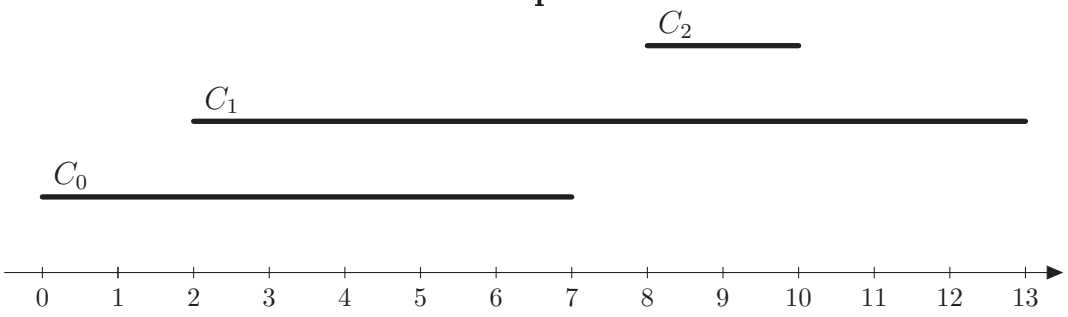
Dans ce problème, on s'intéresse à l'allocation des salles d'un lycée à partir des horaires des cours.

La donnée du problème est une liste de cours. Un cours est simplement représenté par un intervalle $[\texttt{deb}, \texttt{fin}[$, où `deb` est l'instant de début du cours et `fin` est l'instant de fin du cours. Les instants peuvent être décomptés en heures ou en minutes au fil de la journée selon les besoins ; dans ce sujet, les instants sont des nombres entiers et l'unité de temps n'est pas précisée.

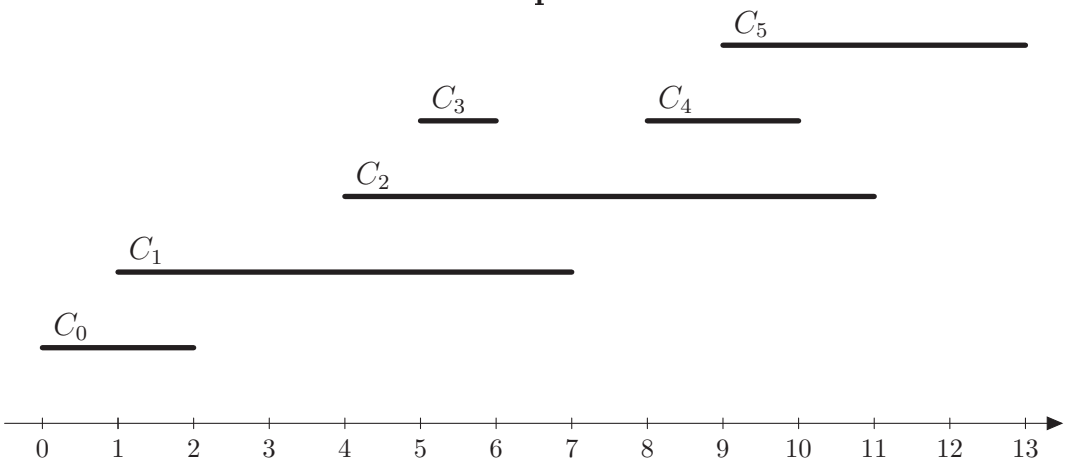
À chaque cours, on doit allouer une salle. Deux cours peuvent se voir allouer la même salle, uniquement si leurs intervalles sont disjoints. En particuliers, deux cours représentés par les intervalles $[4, 6[$ et $[6, 8[$ peuvent se voir allouer la même salle. L'objectif est d'utiliser un minimum de salles.

On présente ci-dessous, deux instances de ce problème.

Exemple 1.



Exemple 2.



Dans l'exemple 1, le cours C_2 est représenté par l'intervalle $[8, 10[$.

I. Pour l'allocation des salles, on parcourt simplement les cours par instants de début croissants et on alloue systématiquement la salle disponible avec le numéro le plus petit.

1. Décrire, sans justification, les allocations ainsi obtenues pour chacun des exemples précédents.

2. Déterminer le nombre minimal de salles nécessaires pour l'allocation dans les deux exemples précédents.

On admet pour la suite que le nombre optimal de salles nécessaire pour une liste de cours est le nombre maximal d'intervalles s'intersectant mutuellement en un instant donné.

II. On s'intéresse dans cette question à une première modélisation du problème. Le cours représenté par la liste Python `[d,f]` se déroule sur l'intervalle mathématique $[d, f[$.

1. Déterminer les listes d'intervalles représentant les exemples 1 et 2.

2. Compléter la définition suivante :

```
def insere(l,elt):
```

où `l` est une liste d'entiers triée dans l'ordre croissant et `elt` est un entier, et qui renvoie une liste triée obtenue par insertion à sa place de `elt` dans `l`. On notera que la liste `l` peut être vide.

On suppose pour la suite que l'on dispose d'une fonction similaire `insereBis` qui opère sur des listes de listes plutôt que sur des listes d'entiers, le critère de tri étant l'ordre des premiers éléments de chaque sous-liste. Plus précisément, la fonction `insereBis` a pour en-tête `def insereBis(LL,li):` et a pour effet d'insérer une liste `li` à la place adéquate dans la liste de listes `LL`.

III. Pour automatiser l'allocation des salles, on va utiliser une autre modélisation. L'intervalle [`deb`, `fin`[représentant le cours numéro i va être représenté par deux événements, modélisés par des listes de longueur 3 : `[deb,i,0]` et `[fin,i,1]`. Une liste d'intervalles pourra ainsi être représentée par une liste d'événements, c'est-à -dire une liste de listes de longueur 3 de la forme `[instant,num,0 ou 1]`.

1. Compléter la définition

```
def traduit(liste_intervalles):
```

qui prend en argument une liste d'intervalles représentés par des listes de longueur 2 et qui renvoie une liste d'événements (listes de longueur 3) correspondante. Notons que pour n intervalles, on obtient $2n$ événements.

2. Pour l'efficacité des algorithmes de résolution, on va travailler sur une liste d'événements triée par instants croissants. On appellera *agenda* toute liste d'événements triés par instants croissants. Quels sont les agendas correspondant aux exemples 1 et 2 ?

3. Compléter la définition

```
def agenda(liste_evt):
```

qui prend en argument une liste d'événements (listes de longueur 3) obtenue par un appel à la fonction traduit et qui renvoie l'agenda correspondant. On pourra utiliser la fonctions `insereBis`.

Tournez la page S.V.P.

IV. On a demandé à des élèves d'écrire une fonction qui vérifie qu'une liste d'événements donnée contient bien autant de fin que de début, et dans le bon ordre, mais sans tester l'appariemment cours par cours, c'est-à -dire sans considérer les numéros d'intervalles.

1. Parmi les quatres solutions proposées, déterminer (sans justifier) la ou les réponses correctes.

```
0  def valideA(agenda):
1      c = 0
2      for e in agenda:
3          if e[2] == 0: c += 1
4          else: c -= 1
5          if c < 0: return False
6          else : return True
7
8  def valideB(agenda):
9      n,c,i,b = len(agenda),0,0,True
10     while b and (i < n):
11         if agenda[i][2] == 0: c += 1
12         else: c -= 1
13         i += 1
14         b = (c >= 0)
15     return c == 0
16
17 def valideC(agenda):
18     for i in range(len(agenda)):
19         if agenda[i][2] == 0:
20             b = False
21             for j in range(i+1,len(agenda)):
22                 if agenda[j][2] == 1: b = True
23             if not b : return(b)
24     return(True)
25
26 def valideD(agenda):
27     c = 0
28     for e in agenda:
29         c += 1 - 2*e[2]
30         if c < 0: return False
31     return c == 0
```

2. Adapter une des fonctions précédentes pour écrire une fonction `intersection_max` qui à partir d'un agenda calcule le nombre maximal d'intervalles qui s'intersectent mutuellement. Justifier la correction de l'approche.

3. En utilisant les fonctions précédentes, écrire une fonction `nbr_optimal` qui à partir d'une liste d'intervalles (modélisation initiale), calcule le nombre de salles nécessaire.

V. **1.** On a demandé à un élève d'écrire une fonction qui, étant donné une liste de booléens, calcule le plus petit entier i tel que la case d'indice i vaille `True`. La fonction renverra -1 si un tel indice n'existe pas. Corriger sa fonction.

```
0  def plus_petit_vrai(liste):
1      n = len(liste)
2      while liste[i] and (i < n) : i+= 1
3      if i = n: return -1
4      else: return i
```

2. On utilise à chaque instant une liste de booléens pour indiquer si une salle est disponible ou non. En utilisant les fonctions précédentes, compléter la fonction suivante qui étant donnée une liste d'intervalles (listes de longueur 2), calcule une liste d'allocations des salles, toujours en allouant la salle disponible avec le plus petit numéro. Dans cette liste, la case d'indice le numéro du cours contient le numéro de la salle allouée.

```
0  def allocation(liste_intervalles):
1      nb_cours = ...
2      liste = ...
3      nb_salles = ...
4      salles_dispos = [True]*nb_salles
5      alloc = [-1]*nb_cours
6      for l in liste:
7          if l[2] == 0 :
8              alloc[l[1]] = ...
9              salles_dispos[...] = False
10         else :
11             salles_dispos[...] = ...
12     return(alloc)
```